SCALING ADDRESS TRANSLATION IN MULTI-CORE ARCHITECTURES USING LOW-LATENCY INTERCONNECTS

A Dissertation
Presented to
The Academic Faculty

By

Vedula Venkata Srikant Bharadwaj

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2017

SCALING ADDRESS TRANSLATION IN MULTI-CORE ARCHITECTURES USING LOW-LATENCY INTERCONNECTS

Approved by:

Dr. Tushar Krishna, Advisor
School of Electrical and Computer Engineering
School of Computer Science (Adjunct)
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Date Approved: December 5, 2017

Dedicated to Amma, Nanna and Anna

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and teachings of my advisor, Prof. Tushar Krishna. I will be indebted to him for his support in both my academics and professional career. His ECE 6100 is responsible for most of what I know in the filed of computer architecture. His availability and accessibility provided me with the right channels to put my points forward. His guidance on both the contents of my work and the way to present them will definitely help me in being a better researcher. I am also thankful to him for advising me on my future career prospects. I look forward to working with him for my future endeavors.

I also extend my sincere gratitude to Prof. Abhishek Bhattacharjee from Rutgers University for his guidance and collaboration during the *NUTRA* project. Prof. Abhishek was always available for discussions and feedback on our research and progress. I am thankful to Prof. Sudhakar Yalamanchili and Prof. Hyesoon Kim for serving on my master's thesis committee and for providing useful feedback on my work throughout the course of my MS.

This thesis also brings my roller-coaster ride of MS to an end. In the last two years, I had the opportunity to interact with many students and forge friendships. Special thanks to Shishir Chawla, Apaar Shanker and Anand Waghmare for the unending technical discussions induced from coffee. Thanks to Ankita Chandak for being there through my ups and downs. Thanks to Anagha Mishra, Gamini Garg and Prakshi Rastogi for believing in my work more than me.

Thanks to *Anna* for supporting me in pursuing my higher studies in the United States and for the absolutely blind support for my work. I am grateful to the unconditional support of my parents, *Amma* and *Nanna* throughout the course of my studies. Here's to another milestone!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

One of the critical operations performed in each memory access in most architectures is address translation. Modern systems employ structures known as Translation Lookaside Buffers(TLB) to accelerate the address translation mechanism. As workloads use ever-increasing memory footprints, TLBs are becoming critical to overall system performance. Modern designs use *private* multi-level TLB hierarchies to balance latency and effective capacity. Unfortunately, private TLB hierarchies have drawbacks, major one being the replication of translations across multiple cores yielding lower hit rates than shared alternatives. But designing scalable *shared* TLBs remains a challenge since the benefit of higher capacity is often outweighed by latency overheads for accessing a large monolithic structure.

To counter the access latencies of large TLBs, physically distributed TLBs akin to NUCA caches can be explored. While a physical distributed last level TLB reduces bank access latency, the on-chip access latency to access remote banks and back continues to hamper performance and energy. Such problems hinder the practical adoption of large shared TLBs on modern many-core systems, where higher core counts exacerbate latency and energy problems.

By utilizing a light-weight single-cycle interconnect based on a recently-demonstrated technique called SMART, this thesis demostrates **NUTRA**, a **N**on-**U**niform **TR**anslation **A**ccess architecture to tackle the scaling challenges of shared distributed last-level TLBs. NUTRA achieves latencies close to those of private L2 TLBs, with hit rates of shared last-level TLBs proposed in previous work. The combination of tight latencies and high hit rates means that NUTRA outperform not only monolithic SLL implementations, but also distributed implementations. Further, this thesis shows that a distributed organization coupled with low-latency interconnects delivers a scalable solution for last level TLBs in multi-core architectures.

CHAPTER 1
INTRODUCTION

Architectures supporting paged virtual memory have been employing Translation Lookaside Buffers(TLB) for accelerating the address translation. Contemporary architectures place the TLB in parallel with first-level caches. Numerous studies showed that TLBs [1, 2] are critical to overall system performance, as they reside on the critical path of memory accesses. TLB misses prompt high-latency page table walks [3, 4] making high TLB hit rates a necessity for good system efficiency. Unfortunately, the advent of big-data workloads with ever-increasing memory needs has historically placed stress on TLB capacities[5, 6].

To meet the demands of the increasing working size of workloads, previous works suggested increasing the size and associativity [7] of the TLBs for higher hit rate in such architectures. Prefetching [4, 6] and superpaging [8] have also been considered for improving the performances of workloads.

Multi-level TLBs: Recognizing the critical role of TLBs in memory accesses, chip vendors have, over the years, responded by realizing larger multi-level TLB hierarchies [9, 10]. Like cache heirarchies, on an L1 TLB miss, the translation is searched for in L2 TLB before triggering a page walk. In many-core architectures, this has been implemented by having private TLB hierarchy for each core. Unfortunately, modern multi-level TLB hierarchies have performance drawbacks. A key problem is that private TLBs on many-core systems suffer from replication of translation entries in TLB structures across multiple cores. This suboptimal use of chip-wide TLB resources is particularly problematic in the face of emerging workloads with big-data memory footprints. Moreover, private TLBs suffer from under- or over-utilization of some TLBs over others depending on the thread running on that core.

Shared Last-Level (SLL) TLBs: Bhattacharjee et al. [11] offer a promising solution

by having a shared last level TLB in multi-core architectures. They have demonstrated 27% higher hit rates compared to private L2 TLBs. However, experiments demonstrated that higher hit rates in SLL TLBs do not directly translate to higher performance, for two reasons. First, with increasing number of cores, the required size of SLL TLB for profitable hit rates increases, increasing the access latency for this structure significantly. Second, more cores reduce the per-core bandwidth at the shared TLB's access ports. With 64 cores, we observe slowdowns of 5-13% when using a monolithic SLL TLB, compared to private L2 TLBs.

One way of tackling this performance challenge is to use a distributed banked implementation of SLL TLBs, rather than a monolithic one. This is similar to NUCA LLCs [12, 13]. However, while this reduces the access latency of each bank, now the on-chip interconnect latency in accessing remote banks dictates performance. Since wire delays do not go down with technology scaling [14], and adding more cores increases the number of on-chip hops, the network latency to a remote bank and back can takes tens of cycles even using state-of-the-art networks-on-chip (NoC) [15], degrading performance. With 64 cores, we observe a 7% slowdown using a distributed SLL TLB versus private L2 TLBs.

Latency is a first-order metric for address translation since it resides on the critical path of a cache access. In this thesis, I present an approach for designing scalable low-latency SLL TLBs called **NUTRA** (**N**on-**U**niform **TR**anslation **A**ccess). NUTRA realizes SLL TLBs as an array of small banks, connected together by a runtime configurable single-cycle sideband network. NUTRA reduces the L1 TLB miss penalty via a three-pronged approach:

- **Higher Capacity**: At its core, NUTRA is an SLL TLB, providing high hit rates due to equitable use of the TLB by all the threads, without any data replication.
- **Low Access Latency**: By having smaller TLB slices distributed across the cores, the access latency of each TLB structure is reduced.
- **Low Network Latency**: NUTRA employs a light-weight interconnect based on the

SMART NoC [16, 17] to connect cores to the distributed TLB slices. This interconnect provides near single-cycle latencies from the source to the remote TLB, reducing network traversal latency.

NUTRA opens up a design-space for distributed TLB architectures, and is similar to NUCA LLCs. However, this work identifies key access characteristics unique to translation accesses and not data accesses, and engineers a solution tailored to that. Overall, this work showcases the following:

- Systems with higher core counts can take advantage of a shared last level TLB because of higher hit rate in multi-threaded applications, but the overall performance degrades because of total latency.
- Workload performance is sensitive to latency of last level TLBs and more than 40% of time, the number of concurrent requests to such an SLL TLB is *one*.
- Distributed implementations of SLL TLBs can decrease access latency but continue to be plagued by high inter-bank hop latencies.
- Employing a low-latency interconnect such as SMART NoC promises close-to-ideal (i.e., single-cycle access latency SLL TLB) performance.
- NUTRA realizes a co-design of distributed SLL TLBs with low-latency network fabric, combining high hit rates and low access latency, and thereby achieving close-to-ideal performance as the number of cores in the system continue to increase.

NUTRA outperforms private TLBs by up to $1.25\times$, at an average of $1.13\times$. NUTRA also outperforms monolithic SLL TLBs [18] and a distributed SLL TLB with traditional NoC. Overall, I show that NUTRA can achieve up to 95% of the maximum ideal speedup.

NUTRA was a collaborative project with *Guilherme Cox* (Rutgers University), Prof. *Abhishek Bhattacharjee* (Rutgers University) and Prof. *Tushar Krishna* (Georgia Institute of Technology).

CHAPTER 2

BACKGROUND AND MOTIVATION

In this chapter we provide an overview of TLB architectures and Interconnects employed in modern systems. We discuss the nuances of various TLB architectures explored previously and the scalability challenges coupled with them. Further, we discuss low-latency interconnect techniques which can be used to scale the address translation. In addition, we also present the related work done in this field.

2.1 Multi-level TLB Hierarchies

Modern systems employ per-core TLB hierarchies for virtual-to-physical translation of instruction and data references. The advent of memory-intensive workloads means that TLBs have become a performance problem, driving research on prefetching techniques [4, 6], superpages [8], etc.

Private Last-Level (PLL) TLBs: All modern systems maintain per-core private multi-level TLBs, as shown in Figure 2.1(a). Private hierarchies naturally suffer from classic problems related to replication of identical virtual-to-physical translations being shared by multiple threads (across cores) running the same parallel program, reducing in overall reduction in capacity [11]. Even with multi-programmed workloads, private L2 TLBs can suffer such replication when accessing shared libraries and OS code. Moreover, multi-programmed workloads suffer from situations where one memory-intensive application may thrash its private TLB while other applications under-utilize their private TLBs [11]. Such sub-optimality in using chip-wide TLB resources implies sub-optimal performance.

Shared Last Level (SLL) TLBs: SLL TLBs have been proposed to solve many of these shortcomings [11]. SLL TLBs increase the hit rate of last level TLB by: (a) possessing a larger number of entries, (b) having no duplication of entries in multi-threaded applications,

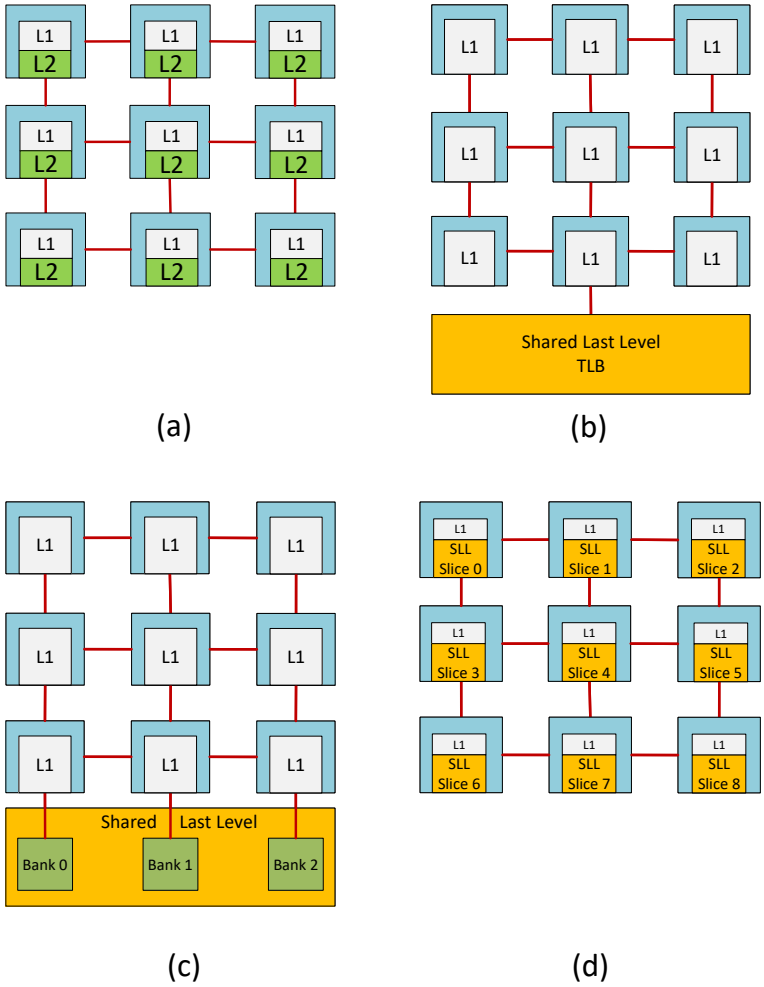

Figure 2.1: *Last Level TLB Organization (a) Private, (b) Shared Last Level TLB - Monolithic, (c) Shared Last Level TLB - Banked, and (d) Shared Last Level TLB - Distributed across the cores*

(c) storing no duplicate global mappings in multi-programmed applications and (d) taking advantage of dynamic TLB partitioning intrinsic to shared structures. Studies on 4-core systems have shown an increase in hit rates by 27% when allocating the cumulative capacity of PLL TLBs to a monolithic SLL TLB [11]. Figure 2.1(b) shows an instance of such a SLL design.

This work found that these hit rates benefits are even more pronounced with more cores. Figure 2.2 shows that an SLL TLB eliminates an average of 61%, 78%, and 86% of the misses suffered by private L2 TLBs on the 16-, 32-, and 64-core systems we model based on Intel Haswell (see our methodology in Section 4). We extract these results on a large-

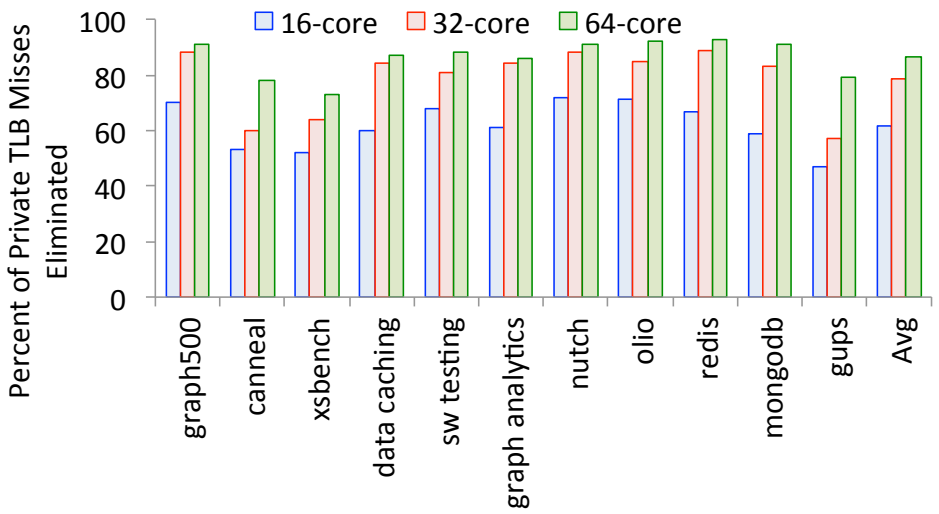

Figure 2.2: *Percent of private L2 TLB misses eliminated when replacing private L2 TLBs with an SLL TLB. Results shown for 16-64-core systems.*

scale system with 2TB of memory, where the inputs to the benchmarks are sized to ensure that the workloads use this memory capacity entirely. We also assume that it is possible to replace all *N 1024-entry* private L2 TLBs with a single SLL TLB of size *N*×*1024*. Workloads with notably poor locality of access (e.g., canneal, gups, and xsbench) are particularly aided by SLL TLBs at even larger core counts as the composite SLL TLB becomes substantially larger than the private L2 TLB. Unfortunately, hit rate reductions do not necessarily translate to performance speedups due to higher access latency and reduced bandwidth in SLLs as we demonstrate next.

2.2 Scalability Challenges with SLL TLBs

SLL TLBs introduce three fundamental scalability challenges. We showcase them using the monolithic SLL in Figure 2.1(b) as an example.

① **SRAM Array:** Scaling the size of any memory array is hard, and SRAMs are no different. We modeled SRAMs in a TSMC 28nm technology node using memory compilers, and plot how the access latency scales as a function of the number of entries in Figure 2.3. All numbers are post-synthesis. We can see that a 1536-entry L2 TLB (the size of Private L2 TLBs in Intel Skylake chips) takes 9 cycles, while a 32×1536-entry design takes close

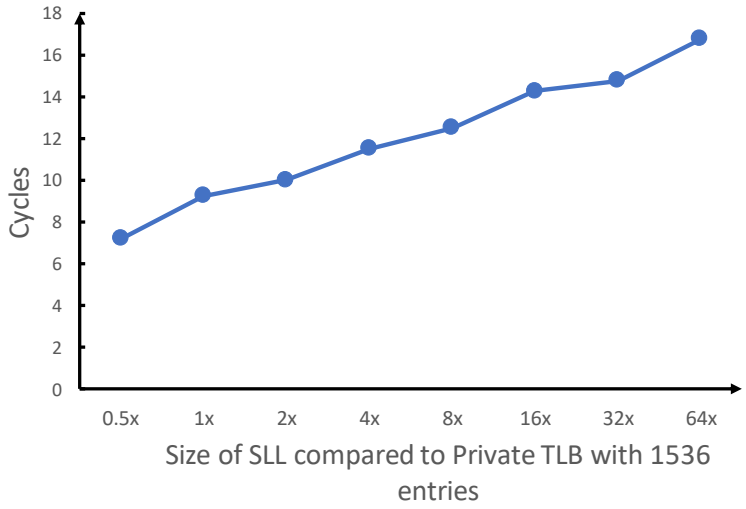

Figure 2.3: *Access latency of SRAM TLB compared to number of entries in a TLB. Post-synthesis in 28nm TSMC PDK.*

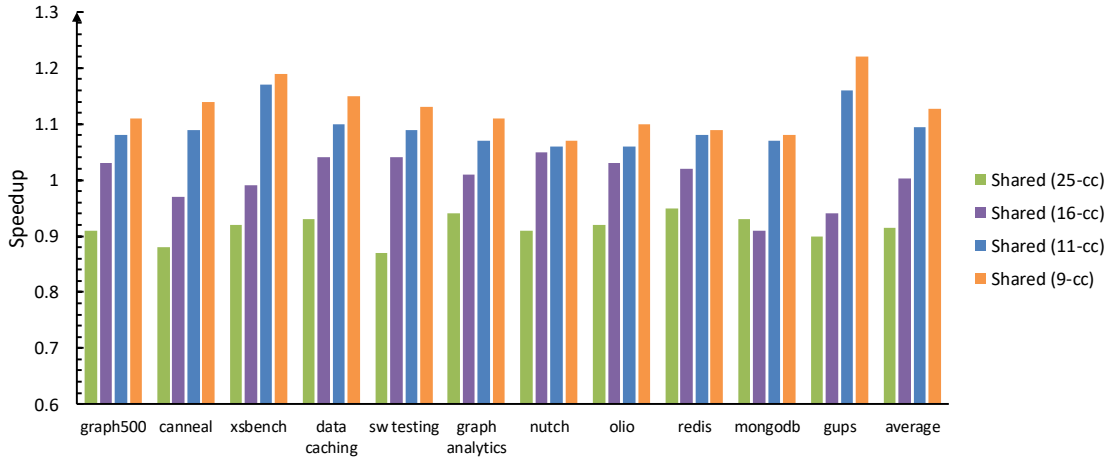

Figure 2.4: *Speedup of workloads with various shared L2 TLB latencies (25-9 cycles) compared to private L2 TLB in a 32-core architecture*

to 15 cycles to access. With future architectures consisting of hundreds of cores, the size of a SLL TLB needed to support the cores would substantially increase, thereby degrading performance of workloads compared to a private L2 TLB architecture.

② **Interconnect:** A monolithic SLL places at one end of the chip introduces additional interconnect delay in accessing the TLB. In a 64-core system, this means that the tiles at the top of the chip would require 8 hops in each direction to access the TLB, increasing latency even further. This would degrade even further at larger core counts.

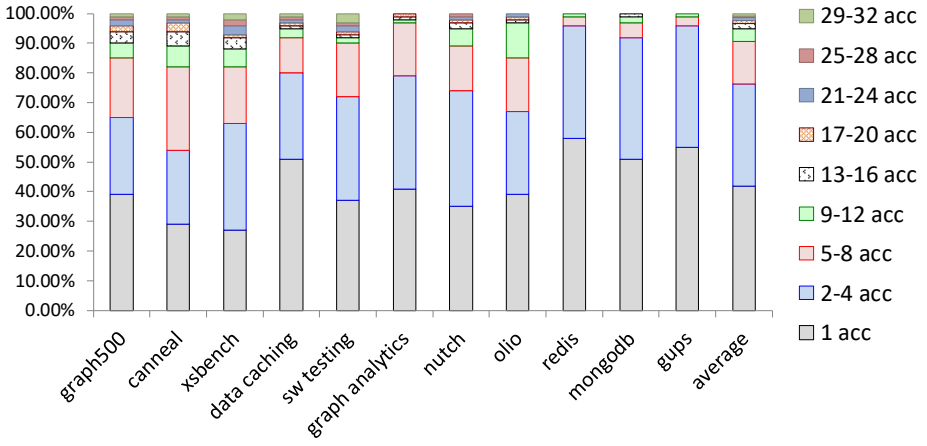

Figure 2.5: *Distribution of number of simultaneous L2 TLB accesses in a 32-core architecture*

③ **Bandwidth:** PLL TLBs enable simultaneous access to each TLB, while monolithic SLL TLBs serialize these accesses, reducing the bandwidth.

Figure 2.4 quantifies the impact of these challenges. We plot the speedup of workloads in a 32-core system with a shared L2 TLB (with 32 times the number of entries in private L2 TLB) having a total latency (access latency + network latency) ranging from 9-25 cycles compared to each core having a private L2 TLB with a 9-cycle latency.

Impact of Latency. Despite better hit rates, not only does the monolithic SLL TLB not improve performance, it actually *reduces* performance by 10-15% when it has an access latency of 25 cycles versus a 9-cycle PLL TLB. Even with an unrealizable *zero*-network latency, the 16-cycle SLL TLB shows little to no speedup over private 9-cycle L2-TLB.

These results show that the workloads are extremely sensitive to the total latency in accessing the shared L2 TLB. Reducing this latency requires changing the organization of the shared structure itself. This shows that there is a need to develop a scalable design for shared last level TLBs in CMPs.

Impact of Banking. Modern LLCs are banked for higher bandwidth. However, extending this approach to SLL TLBs, as shown in Figure 2.1(c), would still require a multi-cycle network traversal each way, limiting the overall performance, as Figure 2.4 highlights.

Impact of Bandwidth. We now attribute these performance issues to latency versus bandwidth. Figure 2.5 shows the percentage of overlapping L2 TLB access events across benchmarks. In other words, every time a core suffers an L1 TLB miss and looks up the SLL L2 TLB, we track how many other L2 TLB accesses are outstanding at that point in time. More than 40% of the time, the SLL TLB is tasked with servicing only one access. This means that access latency, rather the bandwidth into the SLL, is the performance limiter.

Together, these studies show that L1 TLB misses are actually rare. However, when they do occur they are high-latency, hurting performance. This makes the design of a scalable SLL TLB structure different from SLL caches which have higher bandwidth requirements and are more latency tolerant through MLP techniques.

2.3 Low-Latency Interconnects

Since the interconnect determines SLL TLB access latency, opportunities for introducing ultra-low-latency interconnect circuits into the address translation fabric were explored.

On-chip Wire Delay. On-chip wire delay scalability has been an age-old challenge. As technology scales, transistors become faster, but wires do not [14], making wires slower every generation relative to logic. This in fact prompted research into NUCA caches [12, 13]. However, since clock scaling has also plateaued, wire delay in *cycles* remains fairly constant across generations. Long on-chip wires have repeaters at regular intervals, and take about 75-100ps/mm [14, 17, 19]. Thus it is possible to perform a 1-cycle traversal across the chip in modern technology nodes.

Crossbars or High-Radix network-on-chip (NoC). Leveraging the wire-delay argument, one could connect the distributed TLBs with a large crossbar to get a single-cycle connection between any two banks. However, crossbars are known to scale badly in terms of power and area [20], and require expensive arbitration logic that dominate latency [21]. High-Radix topologies like flattened butterfly [22] can also provide dedicated links between

all routers within a row and column. However, they require expensive per-hop multi-ported routers, adding area and power [16]. Moreover, as Section 2.2 demonstrated, the high bandwidth provided by the crossbar and high-radix NoCs is not necessary for TLBs. High-Radix NoCs are expensive and inefficient in reducing access latency between TLB banks.

SMART NoC. SMART [16] is a recent NoC proposal that augments NoC routers with bypass paths, and allows cores to setup single-cycle bypass paths across multiple hops. This can be done either statically [17] at load-time or dynamically [16] at runtime. Each SMART router had an additional mux and arbiter that connects the incoming flit on the link directly to the output link, without sending it to the clocked input queues in the router. This enables a flit to traverse multiple hops in a single cycle, before getting latched at the queue at its destination. In case of contention, it may get buffered at an intermediate router. SMART paths are opportunistic; at low-loads they provide the illusion of dedicated all-to-all single-cycle wires, while at extremely high-loads they are no worse than a baseline hop-by-hop design. The maximum hops that can be traversed in a cycle is known as HPC_{max} and depends on the tile-size, clock-frequency, and wire-delay. SMART has been validated via multiple chip prototypes [17, 23, 19] with HPC_{max} of 8-16 at 1-2 GHz.

SMART is a promising technology for scaling SLL TLBs, since it works best at low network loads, which is exactly the behavior of L1 TLB miss traffic as we observed in Figure 2.5. In this work, SMART was leveraged to design an interconnection fabric between TLBs as I describe further in Section 3.2.4.

2.4 Related Work

The most closely related to this work is by Bhattacharjee et al. [11], which proposes and evaluates a system with shared LL-TLB, similar to our shared monolithic LL-TLB approach. Their work shows the potential of shared LL-TLB without exploring different design choices or the optimization of the interconnect network. They show how shared LL-TLB can reduce TLB miss rate for multi-threaded and multiprogrammed workloads

with a fixed latency LL-TLB lookup. My work focuses on improving the latency of the LL-TLB along with the benefits of having a shared LL-TLB. Synergistic TLBs [31], on the other hand, emulates a ”shared” TLB by migrating and replicating TLB entries across private TLBs running on the same address space. This approach may reduce TLB misses, but underutilized the potential TLB capacity if we have a real shared TLB.

There is a wealthy line of research on reducing TLB miss rate on private TLBs. In Pham et al. [32] design, called CoLT, the system packs translations of contiguous virtual-to-physical mappings into a single TLB entry. The improvement in TLB coverage comes when there exists contiguity and alignment in the pages being mapped. These two requirements can occur naturally or, at a higher degree, with operating system's support. To relax CoLT's requirements, Pham et al. [33] propose a follow-up work that allows groups of contiguous memory regions to use a single TLB entry, removing the need of stringent contiguity within the region. Both of these approaches require simple hardware changes in the TLB, but are limited to few contiguous mappings per TLB entry. To accommodate greater contiguous regions, modern systems [9, 10] use today split-TLBs, or one set-associative TLB for each supported page size. This requires parallel TLB lookups, one for each supported page size, which demand extra energy per lookup. Another option is to lookup one TLB at a time, i.e., one supported page size at a time. This leads to unpredictable turnaround per lookup and poor performance. The research community has proposed alternatives to split-TLB. Seznec [34] leverages the skew cache idea and proposes the same technique for TLBs. Skew TLBs apply different hash functions for each way of the TLB during lookup. Thus, enabling support for multiple page sizes in a set-associative TLB. Cox et al. [35] propose MixTLB, their design improves TLB utilization and TLB miss rate of set-associative TLB that supports multiple page sizes. It does that by coalescing large pages in a single TLB entry. Karakostas et al. [5] propose a TLB design that maps free ranges of contiguous mappings, not constrained by the size of a page or large page. In summary, all these TLB schemes are designed to reduce TLB miss rate or improve TLB coverage, usually

constrained by private area and energy of L2 TLB per core. This work, with SLL TLB could leverage all these techniques, since they are orthogonal to whether the TLB is private or shared.

CHAPTER 3

NUTRA: NON-UNIFORM TRANSLATION ACCESS ARCHITECTURE

The approach presented in this thesis, NUTRA, organizes the SLL TLB as a distributed array of TLB slices connected by a configurable interconnect fabric, as shown in Figure 2.1(d). In this chapter, we discuss the design strategy followed to implement NUTRA and the reason behind the design decisions. We initially discuss the organization of TLB and the interconnect system. Further, the different steps of accessing a last-level TLB in NUTRA is explained in detail.

3.1 TLB Organization: Distributed TLB slices

The overall organization of NUTRA is a logically shared last level TLB distributed across the tiles of a many-core system. It mirrors the design of NUCA LLCs [12]. Each slice is the same size or smaller than the size of current PLL TLBs, thereby meeting the same area and power budgets.

TLB Entries: Each entry in the slice includes a valid bit, the translation and a context id associated with the translation.

Indexing: Although optimized indexing mechanisms can be adopted for better performance, we use a simple indexing mechanism using the virtual address as shown in Figure 3.1. We expect future studies to build on our work and achieve even better performance

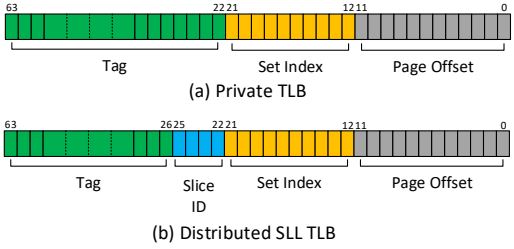

Figure 3.1: *Composition of a virtual address for a (a) Private TLB; and (b) Distributed SLL TLB with 1024-entry slices.*

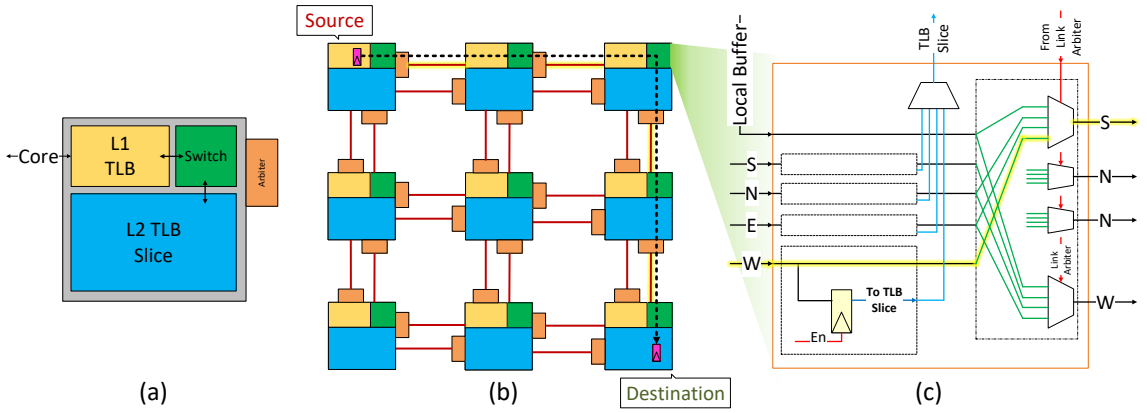

Figure 3.2: *(a) TLB hierarchy present near each core in NUTRA. (b) Source and destination of a request and the path of taken by the request. (c) Micro-architecture of the switch which enables single cycle traversal through the network.*

with better indexing functions.

3.2 TLB Interconnect

The advent of many-core systems has prompted significant research on interconnection networks. Cores are connected through scalable interconnection networks for communication and synchronization. A physically distributed SLL TLB would also need an interconnect system for communicating between the cores and TLB slices. We propose a dedicated side band interconnection fabric for sending messages involved in access and control of a distributed shared last level TLB. Further details about the message types and interconnect bus width are discussed in Section 3.3.

While a dedicated interconnect fabric for translations can simplify the complexity involved in sending requests and responses involved, it is difficult to break the latency barrier of multi-cycle hops involved in a traditional multi-core network to reach a remote slice. Consider the case of a 16-core system connected through a mesh topology, a translation request can take anywhere between 2 to 12 cycles for traversing to a remote slice. Even with high radix routers that can have dedicated links, there is a prohibitively huge area and power cost. In a distributed SLL TLB environment, this latency can easily dominate the total access latency and nullify or overshadow the performance gains from the high hit rate.

3.2.1 Datapath: Bufferless SMART Network

SMART [16, 17] is promising for scaling SLL TLBs as it works best at low network loads, which is exactly the behavior of L1 TLB miss traffic (see Figure 2.5). However, there are two challenges with directly adopting SMART:

(i) SMART was proposed for general-purpose NoCs carrying cache lines. Introducing a full SMART NoC between SLL TLBs with buffered routers and crossbars at every hop is expensive in terms of area and power.

(ii) SMART paths are opportunistic and steal bandwidth when available. However, L1 TLB misses are latency-sensitive and having guaranteed bypass paths is preferable. Moreover, every router needs to manage arbitration and buffering for requests that win a partial path, adding complexity.

In NUTRA, we leverage the underlying mechanism of SMART of single-cycle traversals across multiple-hops, but redesign the datapath to be low-cost.

We add a simple *bufferless switch* next to each L2 TLB slice to either latch an incoming message and send it to the L2 TLB request queue, or connect it to any of the output ports. We show this design in Figure 3.2. Requests to a remote SLL TLB slice involve sending a request message from the requesting node to the destination node over a circuit-switched single-cycle path through these switches, as Figure 3.2(b) highlights. Figure 3.2(a) shows the switches present at each core in a distributed SLL TLB system in 16 core mesh based architecture. Each switch has muxes for every output port, to connect an input port to either an output port, or to the local TLB slice. This is preset by a fine-grained circuit-switching mechanism that we describe in Section 3.2.2. Each traversal over this network takes a single-cycle[1]. This network is used by both requests and responses.

[1]For large chips running at very high frequencies, this might be multiple cycles by adding pipeline latches as we discuss in Section 3.2.4

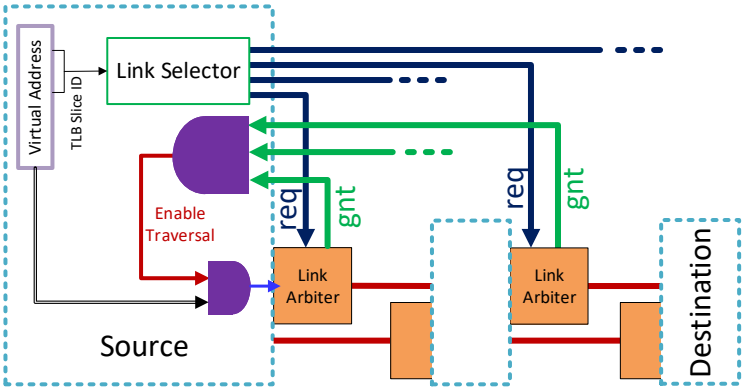

Figure 3.3: *For setting up the path a core sends requests to all link arbiters in the path and waits for grants from them.*

3.2.2 Control Path: Fine-Grained Circuit-Switching

We now describe the various steps involved in sending the messages involved in NUTRA.

Path Setup: For each traversal through the interconnect, links in the path have to be acquired before sending any kind of message. To ensure that the packet reaches the destination in a single cycle, *all* links in the path must be acquired in the same cycle. This is done over a separate control wires. Each link has an arbiter which can allocate the link to one of the requesting cores. Figure 3.3 shows an example of a core sending *requests* to all link arbiters in its path and receiving *grants* from each link arbiter before traversing the path. If any requester fails to acquire *all the links* in the desired path, because of any contention, it will wait and try again in the next cycle. This ensures that there are no packets traversing partial paths and thus avoids complexity. Figure 3.2(b) shows an example of a traversal across the chip through the TLB interconnect fabric.

Control Wires: Each core has must have a way to setup a path to any of the slice present in the system. The width of the control wires directly depends on the routing policy adopted by the TLB system at design time. Consider an XY based policy in a system as shown in Figure 3.2(c). Each core is connected to the arbiter associated with a link through which the core can send a request. Thus, the number of wires going out of each core is $(num\_cores\_in\_each\_row - 1) \times (num\_rows - 1) \times (num\_columns)$. For a 16-core mesh architecture, this equals 15 bits from each core. Each of the arbiter has another wire

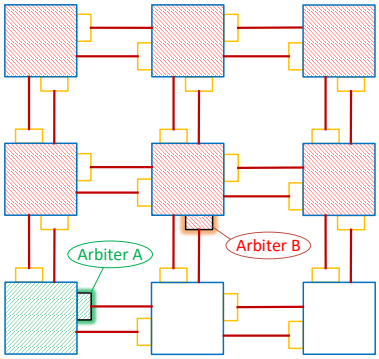

Figure 3.4: *Arbiter complexity: Cores that can send requests to a given arbiter*

to send the grant signal. Thus would be total of 30 bits of control wire per core in a 16 core mesh architecture.

Link Arbiters: Each network link has an associated arbiter residing near the switch. The arbiter can get requests from any core which can send a TLB request/response packet through the link. This arbiter then selects one of the requesting cores and grant the link to it for the next cycle by setting the output mux to receive from the right input port, and sending a 1-bit acknowledgment back to the requester.

Depending on the routing policy, not all the cores can send requests to any given arbiter. Figure 3.4 shows the case where the packets can only traverse by following an XY routing policy. Arbiter A can only receive requests from the (green) switches in present in the horizontal row as itself. On the other hand, Arbiter B needs to take care of requests from all the (red) switches present North of itself. This optimization reduces the number of requests an arbiter has to serve, thereby reducing its area and power. The trade-off with this is reduced bandwidth in the network because of the XY limitation. In contrast, a routing policy that say chooses randomly between XY and YX will lead to lesser contention between requesting nodes, but require larger arbiters.

3.2.3 Switch Microarchitecture

Figure 3.2(b) shows the micro-architecture of each switch in NUTRA. The link arbiters send the select signals for each direction in the network, which will be consumed by multiplexers present inside the switch to select one of the incoming packets from different di-

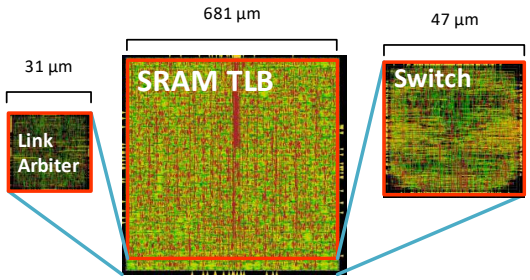

Figure 3.5: *Place-and-Routed NUTRA tile in 28nm TSMC with the L2 TLB SRAM, switch and link arbiters highlighted.*

rections. Figure 3.2(b) shows how a request arriving from the *West* direction traverses the switch and propagates further in the *South* direction as selected by the multiplexers. This form of circuit switching therefore enables single cycle traversal throughout the network.

When the message reaches the destination switch, it is latched onto a buffer, from where it can access the TLB slice present there. The switch shown in the figure allows any kind of turns to take place. An optimized switch would contain only the turns possible for the routing policy employed by the TLB interconnect system. Since the switches consist of only basic multiplexers, they consume minimal area/power.

3.2.4 Implementation

We implemented the NUTRA interconnect in TSMC 28nm with a 2GHz clock. Figure 3.5 shows the place-and-routed design. We highlight our observed insights here.

Critical Path. There are two sets of critical paths in the interconnect.

- *Datapath.* On the datapath, a multi-hop traversal through all the intermediate switches needs to be performed within one clock cycle. Recall that the TLB interconnect is created at design-time. If timing is not met at the desired clock frequency, pipelined latches can be added at the maximum hops per cycle (HPC_{max}) [16] boundaries. This will increase the network traversal delay, but does not affect the operation of the design.
- *Control Path.* On the control path, the critical path consists of the path setup which includes sending a request to the furthest link arbiter, link arbitration, and the grant

Table 3.1: Power and area of a switch and link arbiters for each slice in comparison to a SRAM based TLB slice. Technology = 28nm TSMC. Target Clock Period = 0.5ns

Per Core	Power(μW)	Area(μm^2)
Switch	430	2209
Link Arbiters	2397	961
TLB SRAM	4022	464635

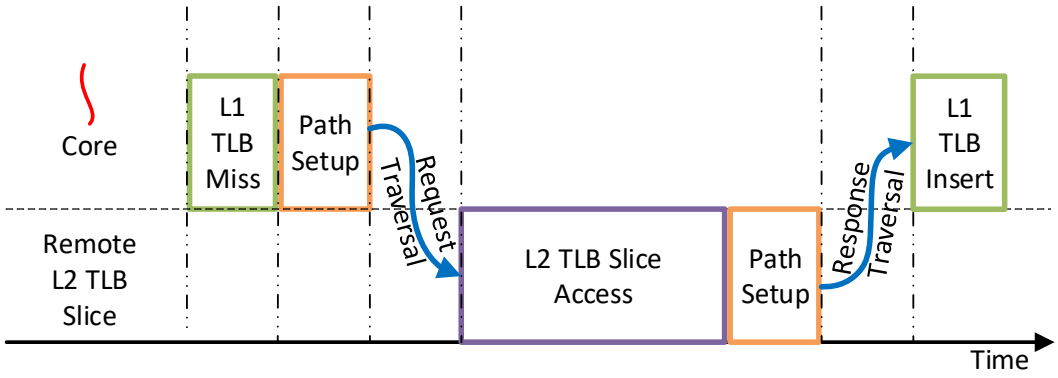

Figure 3.6: *Timeline of a virtual address translation in case of an L1 TLB miss and remote L2 TLB access in NUTRA*

traversal back to the core (Figure 3.3). We observed that the place-and-route tool placed all the arbiters close to the center of the design to reduce the average wire lengths to meet timing.

Area and Power. Table 3.1 shows the post-synthesis power and area consumed by the TLB interconnect switch and arbiter. We contrast it with the cost of the L2 TLB SRAM present in the same tile. The area consumed by switch and arbiter is negligible compared to the tile's L2 TLB SRAM. In fact, the power consumed by the arbiters is comparable to that of the SRAM access, increasing the power consumed on remote TLB accesses. However, we still save power overall, as we demonstrate later in chapter 5.

3.3 Timeline of L2 TLB Access in NUTRA

Figure 3.6 presents a timeline of a virtual address translation when there is an L1 TLB miss

- **L1 TLB Miss.** The L1 TLB miss triggers a circuit-switched path setup. The path setup can be done speculatively during the L1 TLB access as well.

- **Request Path Setup.** The remote TLB slice to which the translation is mapped is identified using the breakdown shown in Figure 3.1. A path setup request is then sent to all the link arbiters along the XY route. The grants from all the requests are ANDed to determine if the full path was granted or not. If not, the path setup is retried. If the full path is granted, the request is sent out.

- **Request Traversal.** The TLB request is forwarded to the switch connected to the TLB slice (Figure 3.2(a)). No header or routing information needs to be appended, since the path is already setup. The request takes a single-cycle through all the intermediate switches, and is latched at the remote TLB slice and enqueued into its request queue. A timeout at the requesting core ensures that any unfulfilled requests are sent again.

- **L2 TLB Slice Access.** The remote TLB slice receives the request and services the request. The translation may either exist or not. If it is a TLB hit, a response should be sent. The response contains the physical page associated with the virtual address in the request. A TLB miss would lead to a page walk which is discussed in section 3.6.

- **Response Path Setup.** A circuit-switched path for the response is then setup by the remote TLB slice.

- **Response Traversal.** The response traverses the TLB interconnect within a single-cycle.

- **L1 TLB Insert.** The requested translation is inserted into the requesting L1 TLB.

3.4 L2 TLB Access Latency and Energy

We quantify the benefits in latency and energy that NUTRA provides over a Monolithic and Distributed SLL TLB.

Figure 3.7 shows the latency of a message when traversing different number of hops through the TLB interconnect in the different SLL TLB designs. We consider two cases.

Case 1: The requested translation is indexed in the slice present in the requesting core: The virtual address is used to index into the SLL slice in the local node and the translation is returned to L1 TLB. The total latency incurred is equal to $access\_latency$ of the TLB slice for both Distributed and NUTRA designs. This is identical to a PLL TLB latency.

Case 2: The requested translation indexes to a remote slice: The required translation request is sent to the remote node containing the slice through a dedicated network. Once it reaches the destination node, the virtual address is used to index into the SLL slice and the translation is then sent back to the requesting slice. Upon receiving the translation response, the requesting core can then forward the translation to the L1 TLB. The total latency in this case is $acesss\_latency + network\_latency$. Here, NUTRA provides a tremendous latency advantage over both Monolithic and Distributed. Even when the maximum hops per cycle HPC_{max} in NUTRA goes up, it is still much faster than Distributed.

Figure3.8 shows the energy consumed by a message when traversing different number of hops through the TLB interconnect to understand the trade-off space of the SLL designs. Most of the energy savings for distributed and NUTRA come from accessing a smaller SRAM structure than a monolithic(M) SLL TLB. Further, on the datapath, because of circuit switching, the energy consumed by an intermediate switch in NUTRA(N) is less compared to a switch in a traditional distributed network(D) with multi-cycle hops. However, NUTRA has a more expensive control path because of multiple request and grant wires spanning to all the link arbiters for simultaneous arbitration (Figure 3.3). For instance, to traverse 14 hops within a cycle, NUTRA will require 14 links to be arbitrated for simultaneously. This shows up as a slightly higher control cost than Distributed. However, the latency gained because of this approach leads to an overall energy savings, as we discuss in Section 5.3.

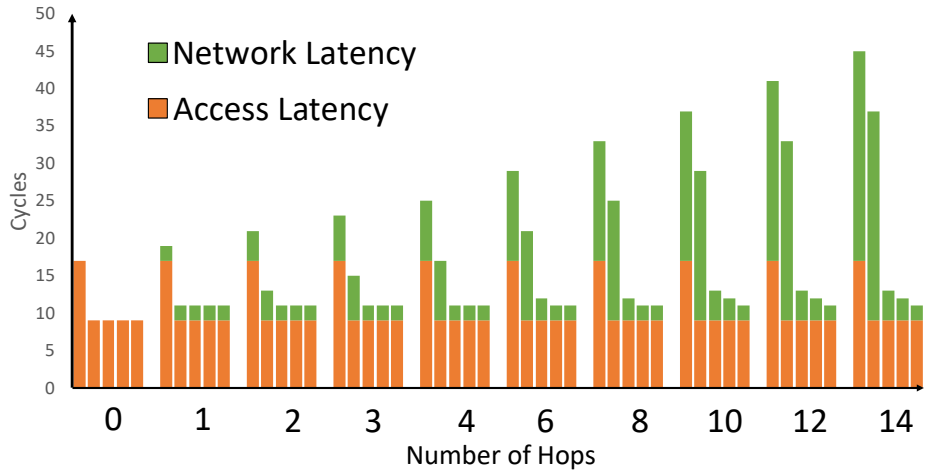

Figure 3.7: *Latency of each message in the TLB Interconnect in various configurations from left to right : Monolithic with Multi-Cycle hops, Distributed with Multi-Cycle hops, and NUTRA with $HPC_{max}4$, NUTRA-$HPC_{max}8$, NUTRA-$HPC_{max}16$*

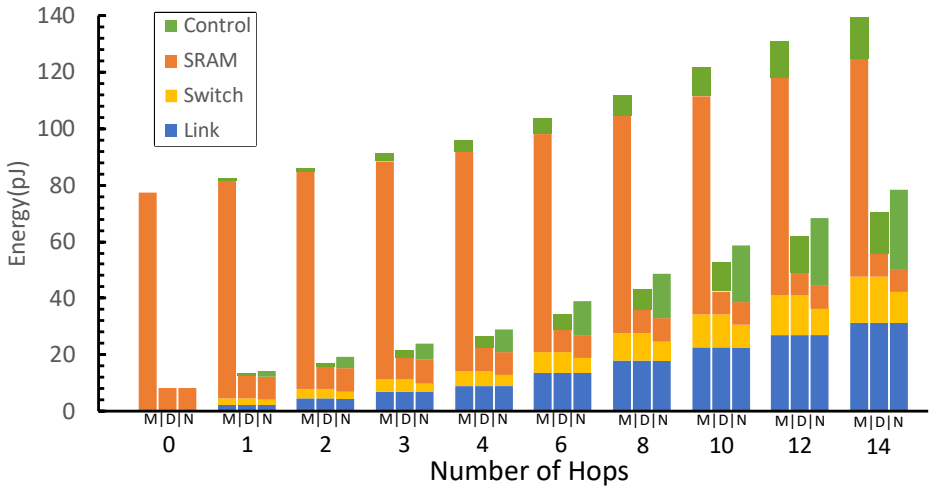

Figure 3.8: *Energy consumed by each message in the TLB Interconnect in various configurations (M)onolithic, (D)istributed, and (N)UTRA vs number of hops*

3.5 Insertion/Replacement Policy

A logically shared TLB promotes inter-core sharing of entries and to support that we use a *mostly-inclusive* policy between L1 and SLL TLB slices. This ensures that a tight coordination between L1 and the L2 slices is not needed.

3.6 Handling Page Table Walks

Suppose that a core suffers an L1 TLB miss and must look up the SLL L2 TLB. Suppose further that it determines that the TLB slice housing the desired translation lies on a remote node. If lookup of the remote node's TLB slice ultimately results in a miss, there are two options for performing the resulting page table walk. In the first option, the remote slice can send a *miss* message back to the requester node, which must now perform the page table walk. In the second option, the remote node can itself perform the page table walk. Both approaches have pros and cons associated with them. Handling page table walks at the remote node is attractive in that it eliminates the need for a *miss* message to be relayed between the remote and requester nodes. However, handling page table walks on the remote node also increases the potential for page table walker congestion; i.e., if multiple core's send requests to a particular remote slice and all of them miss, page table walks can be queued up. We evaluate both these options in Section 5.

3.7 TLB Shootdowns

A key design consideration involves how NUTRA responds to virtual memory operations performed by the OS. In particular, consider a situation where a page table entry is modified by the OS on a particular core. When this happens, the OS kernel usually launches inter-processor interrupts (IPIs) that pause other cores and run an interrupt handler that "shoots down" or invalidates the stale translation in the TLB. This operation requires care in NUTRA – specifically, it is now possible that multiple cores simultaneously relay a translation invalidation signals to a single TLB slice that houses the stale translation. This can quickly congest the system by cascading TLB invalidation lookups of a single TLB slice.

We sidestep this problem by designating certain node(s) as the *invalidation leader*. In other words, even though *any* core can receive IPIs as usual, and each core invalidates its private L1 TLB, only specific cores are permitted to then go further and relay invalidation

signals to the SLL TLB. For example, if core 0 is considered the invalidation leader, *any core* that receives an IPI has to relay a message to core 0. Core 0 in turn relays a message to the relevant SLL TLB slice to invalidate the stale translation. We study the performance impact of this approach, varying the number of leaders in the system in Section 5. The ideal scenario is a middle ground where the number of leaders is far fewer than the total core count, but where it is not so small that the messages become congested at any particular leader core.

CHAPTER 4
METHODOLOGY

In this chapter we provide the evaluation methodology and the workloads used to evaluate NUTRA.

To justify taking distributed SLL TLBs, we mainly focus on the total performance speedup of workloads in multi-core environment. We take Intel Haswell [24] based multi-core configuration which include different L1 D-TLBs for each page size and a common L2 TLB. Although NUTRA benefits both I-TLB and D-TLB performance, we focus on D-TLBs because of its greater performance impact.

4.1 Simulation framework

We use a Simics-based simulation framework to model 16-64-core Haswell systems with 32KB 8-way L1 instruction and data caches, 256KB 8-way L2 caches, and an LLC with 8MB per core. The system uses 2TB of RAM, and runs Ubuntu Linux 4.14 kernel series. Both the hardware and OS can support transparent superpages as is standard. Furthermore, each core maintains 64-entry L1 TLBs for 4KB pages, 32-entry L1 TLBs for 2MB pages. Our baseline design without NUTRA uses per-core private 1024-entry TLBs that can concurrently support translations for 4KB and 2MB pages. We evaluate three NUTRA L2 TLB designs (fixing the total size) as shown in Table 4.1.

For this study, we consider a mesh based topology for the multicore configuration. Since SMART [16] can be used on any kind of topology, NUTRA can also be extended to various topologies for similar performance benefits. Further, we consider two different types of TLB interconnect system for shared L2 TLB organizations: (a) *Traditional Multi Hop:* This configuration involves traditional 1-cycle router coupled with 1-cycle link latency. To compete against SMART-based NUTRA, we assume there are enough buffers

Table 4.1: Major configurations of TLB that were simulated.

	L2 TLB Entries	Physical Org	Interconnect
Private	1024	1 TLB Per Core	-
Monolithic (Shared)	1024×NumCores	Monolithic	Multi-Cycle Hops
Distributed (Shared)	1024×NumCores	1 slice Per Core	Multi-Cycle Hops
NUTRA	1024×NumCores	1 slice Per Core	SMART

and links in the system to not have any kind of link contention in this network. Including any network contention may degrade performance of workloads further for Traditional Multi-Hop Networks. (b) *SMART Interconnects:* A single cycle traversal if there is no contention; otherwise waits for another cycle as explained in Section 3.2.2.The interconnects follow an XY based routing mechanism, which means that the messages first traverse in the X-direction and then in the Y-direction.

4.2 Benchmark suite

We use a wide set of benchmarks from Parsec [1] and CloudSuite [25] that have non-negligible TLB miss overheads to evaluate the various configurations. The inputs were scaled up to utilize upto 2 *TB* of memory. Further, we study the performance of multi-programmed workloads in such a distributed SLL system. We take combinations of 4 applications and evaluate any performance degradation. Each application in a multi-programmed workload has 8 threads executing and scaled up to use 2TB of memory. We modify benchmark inputs to achieve this memory scaling.

CHAPTER 5

EVALUATIONS

We now showcase the results obtained from our evaluations of NUTRA. We first look at the performance and power implications of NUTRA followed by an anlaysis of the interconnect system. In addition, we also summarize our observations from the multi-program workloads.

5.1 Performance

We begin by studying the performance of NUTRA on a 16-core system. Figure 5.1 shows the speedups that NUTRA provides versus a baseline Haswell system with private L2 TLBs. The higher the speedup number the better. For reference, we compare NUTRA with a MONOLITHIC SLL L2 TLB with realistic network and access latencies from our circuit-level design study in chapter 4, a DISTRIBUTED SLL L2 TLB that improves over the MONOLITHIC case, and an IDEAL configuration where all SLL TLB accesses are single-cycle. Note that IDEAL does *not* imply an infinite TLB; i.e., it is purely a way to identify how well we can do with the best possible access latency. Finally, Figure 5.1 focuses on a scenario where Linux generates only 4KB pages.

Figure 5.1 shows that NUTRA can achieve speedups versus PLL TLBs that are as high as 1.25× with an average speedup of 1.13× across all benchmarks. Moreover, NUTRA consistently outperforms all other configurations. In fact, MONOLITHIC suffers from a net performance loss versus PLL L2 TLBs because of the higher access latency. While DISTRIBUTED approaches partly help, NUTRA easily achieves over 8% additional performance benefits and comes within 2% of the ideal performance.

Figure 5.1 shows the potentially more interesting case where Linux's native support for transparent 2MB superpages is turned on. For each workload, we found that the OS

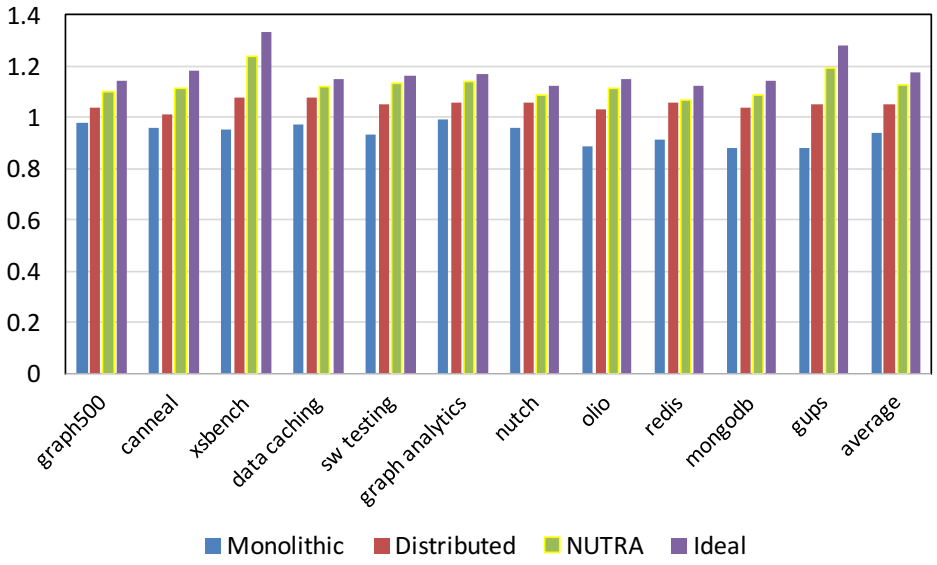

Figure 5.1: *Speedup of workloads in various configurations compared in a 16 core architecture with 4KB pages compared to private L2 TLB*

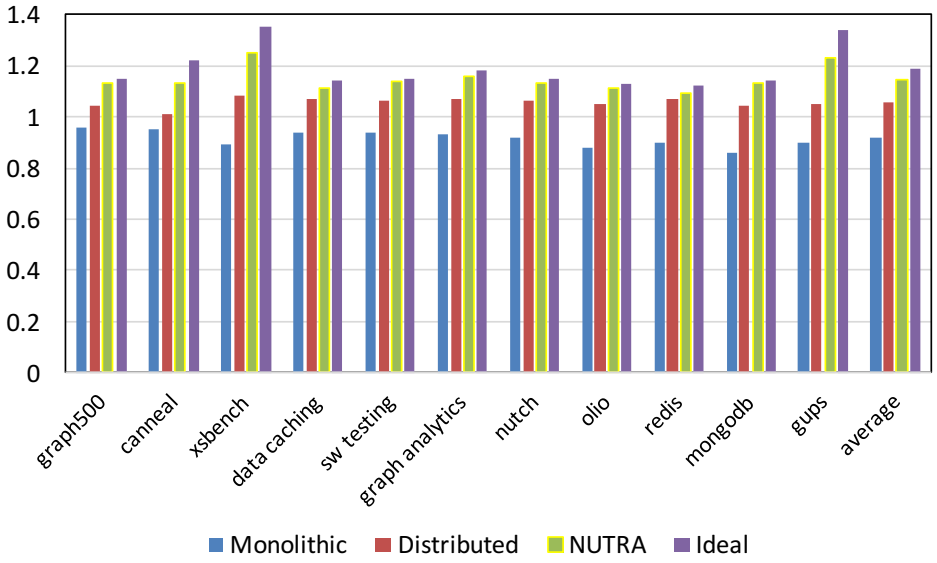

Figure 5.2: *Speedup of workloads in various configurations compared in a 16 core architecture with THP compared to private L2 TLB*

was able to allocate roughly 50-80% of its memory footprint with superpages. One might initially expect superpages to reduce L1 TLB misses to the extent that the benefits of NUTRA recede. In reality, however, we achieve *even better* performance with NUTRA in the presence of superpages. This is because our workloads demand so much memory (i.e., 2TB) and require so many 2MB superpages, they continue suffering from L1 TLB misses.

Therefore, the SLL L2 TLB continues to be accessed, making its hit rate and access latency critical determinants of overall system performance. This is coupled with the fact that superpages do however eliminate expensive page table walks – i.e., many SLL L2 TLB accesses become hits. But this means that SLL L2 TLB access time becomes even more critical because long-latency page table walks are reduced, making SLL L2 TLB performance a bigger contributor to overall performance. This explains why workloads such as *xsbench* and *gups* achieve large speedups of 1.2×+. Furthermore, NUTRA outperforms monolithic and distributed with even larger margins than when simply using 4KB pages.

5.2 Scalability

Next, we look at the performance impact when the number of cores increase. Figure 5.3 shows the speedup of NUTRA and other configurations compared a PLL TLB baseline for 16-,32-, and 64-core systems Linux supper for superpages turned on. We show average, minimum, and maximum speedup numbers. The higher hit rate offered by a SLL TLB is overshadowed by the high access latency in a MONOLITHIC configuration. Employing a sc distributed TLB with a traditional network leads to a performance improvement but it is far more modest as compared to NUTRA. In fact, NUTRA achieves somewhat higher performance benefits at higher core counts as the aggregate SLL L2 TLB becomes bigger.

5.3 Energy

Recent work shows that address translation can constitute as much as 10-15% of overall processor power and that the energy spent accessing hardware caches for page table walks is orders of magnitude more expensive than the energy spent on TLB accesses [26]. Naturally, using an SLL TLB therefore saves address translation energy by eliminating a massive fraction of page table walks due its higher hit rate. Figure 5.4 shows this, by plotting the percent of energy saved versus a baseline with PLL TLBs. Even the MONOLITHIC approach eliminates roughly a third of address translation energy. However, NUTRA elim-

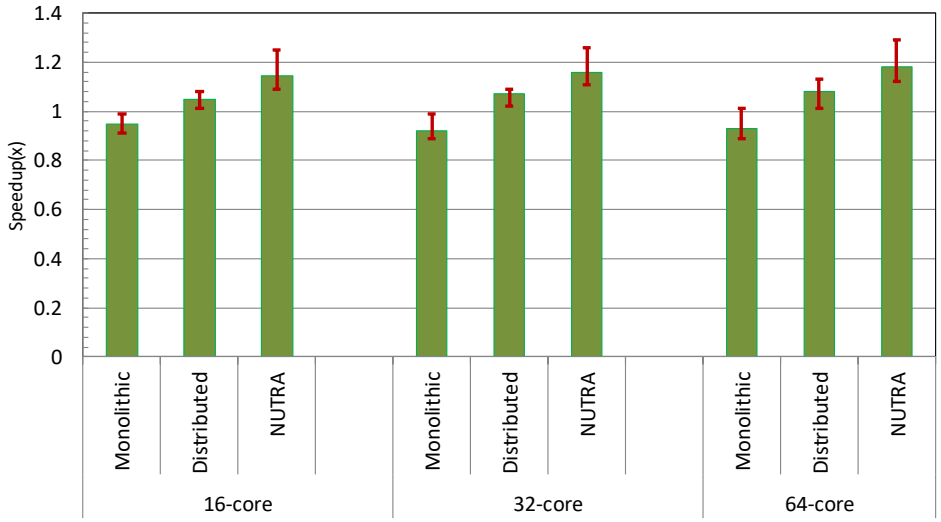

Figure 5.3: *Speedup of configs in various multicore architectures with THP compared to private L2 TLB system*

inates even more energy (as much as 60% on 64 cores) by dramatically shortening runtime and therefore eliminating static energy contributions to address translation energy. These benefits are achieved despite the energy overheads of the dedicated network.

5.4 Interconnect system

We now study our SMART NoC-based interconnect. First, we compare NUTRA to a traditional multi-cycle network and an ideal interconnect-based distributed SLL. The ideal interconnect has a perfect interconnect with 0-cycle network latency without any contention. Figure 5.2 shows the speedup of workloads using NUTRA, ideal and traditional network compared to a private L2 TLB. NUTRA outperforms the traditional network and delivers near-ideal speedup.

Effect of Contention: We also implemented an ideal NUTRA with a contention-free interconnect. Figure 5.5 shows the effect of contention in the implemented NUTRA compared to both contention-free ideal-NUTRA system and the ideal interconnect with 0-cycle latency. Although SMART delivers a 1-cycle traversal only when there is no contention, this result shows that the effect of contention is minimal because of the pattern of L2 TLB requests.

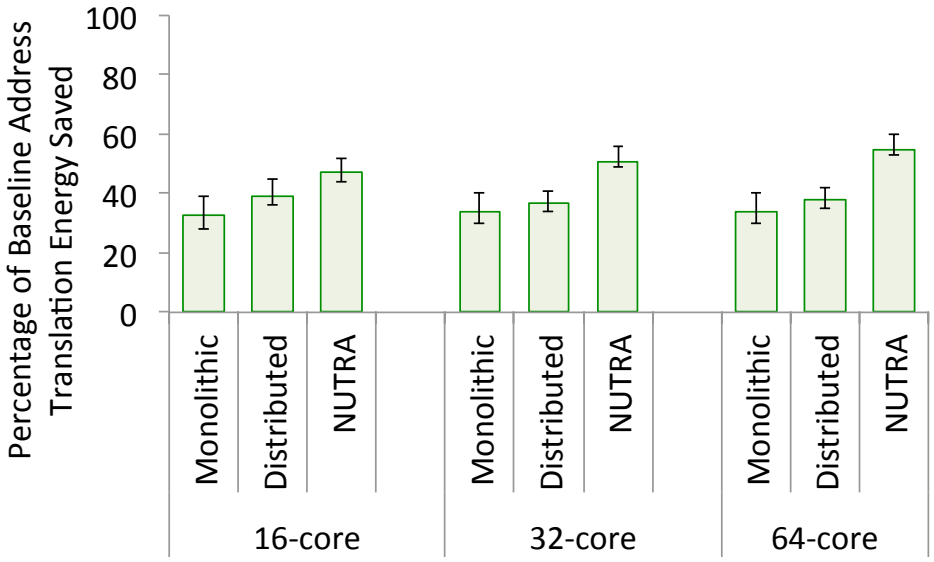

Figure 5.4: *Percent of baseline energy used for address translation in a system with PLL TLBs saved by using* NUTRA *versus other approaches.*

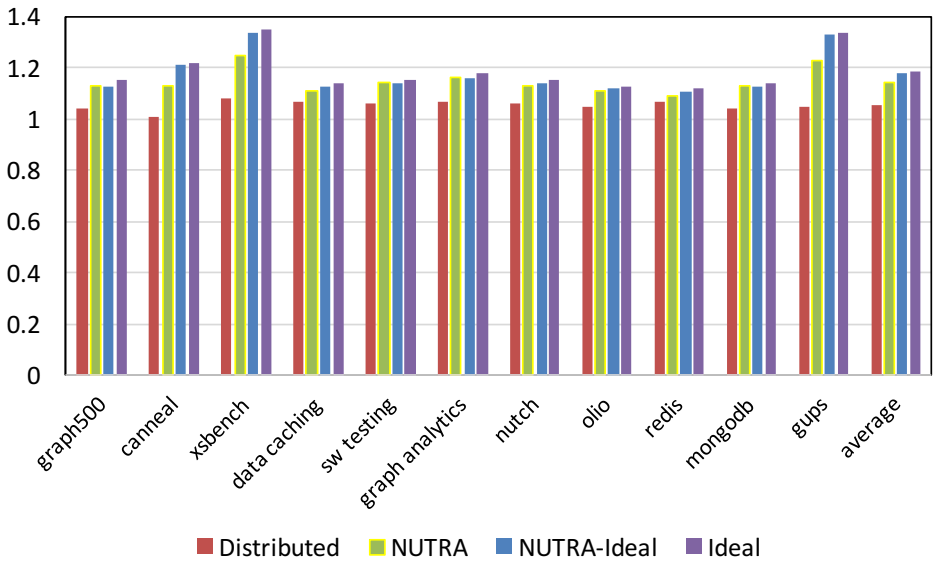

Figure 5.5: *Speedup of our NUTRA implementation, an Ideal NUTRA system and a zero-network latency distributed TLB configuration compared to PLL TLB*

Co-design: To see the effect of having a low-latency interconnect, we implemented a configuration with a monolithic SLL TLB and SMART interconnect connecting it to cores. Figure 5.6 shows that having just a low-latency interconnect is not enough, instead a co-design of distributed TLB along with a low-latency interconnect is required for performance gains. A monolithic SLL TLB with SMART (Monolithic+SMART) degrades in performance versus PLL TLBs.

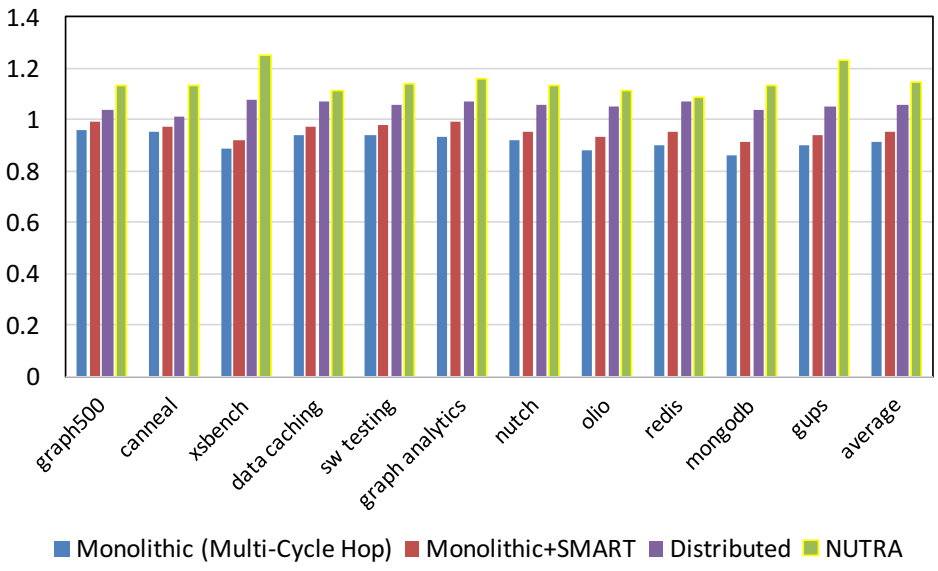

Figure 5.6: *Speedup of our NUTRA implementation, and a monolithic SLL TLB with SMART network configuration compared to PLL TLB*

Average Latency: Ideally messages in NUTRA should only take 1 cycle to traverse, but because of any contention messages wait for consecutive cycles to get access to links through which they want to traverse. We find that such latencies are generally between 1-3 cycles. We only found two workloads – xsbench and gups – with such high TLB miss rates that network congestion became a bigger issue, causing latencies to increase to 3-6 cycles.

Path Setup: We study two modes of link reservation: (a) Round trip Acquire: Links would be acquired for the total period of accessing a remote slice. In this mode, link selection has to be performed only once for sending a request and response. (b) One Way Acquire: Links are acquired only for sending a one-way message. Each message in the system has to perform the link selection before traversing. Figure 5.7 shows the performance of workloads comparing the two modes of link selection. We found out that acquiring the links separately for each message delivered better performance than acquiring links for round trips.

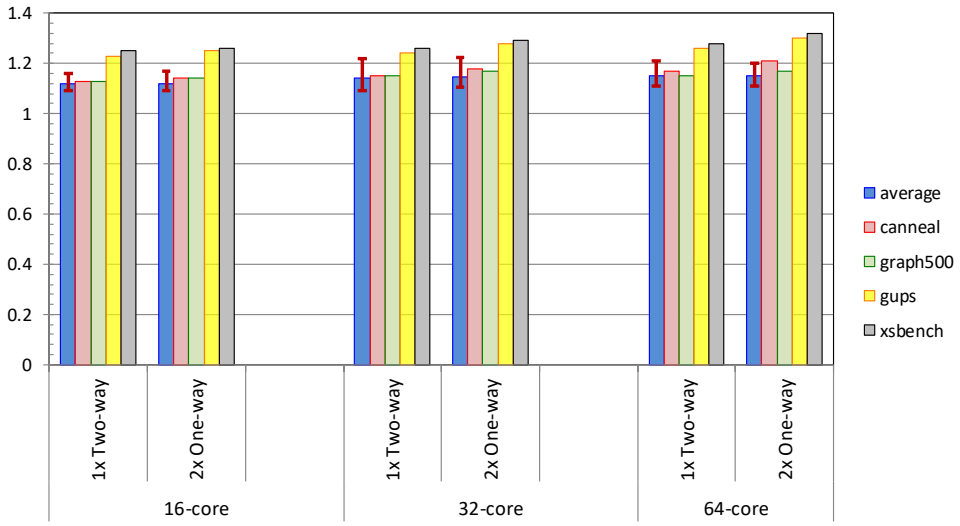

Figure 5.7: *Speedup of workloads comparing two different types of link acquisition policies*

5.5 TLB Invalidation

We investigated the effect of sending an invalidate request to a TLB slice because of shoot-down or flush from any core. We considered various ways in which an invalidate message can be sent across a the TLB interconnect. The straightforward way is to send an invalidate from each core to the TLB slice. This policy is simple but may lead to congestion in the interconnect if all the cores are trying to invalidate from the same slice. The other way is to send the invalidate message to a central location which can then manage invalidations to all the slices. This can be further split up by having a manager for a set of n slices. Figure 5.8 shows the speedup of workloads with different ways of sending an invalidate message compared to each core sending its own invalidate message. It can be seen that the optimal way is to have a TLB manager per 4 cores.

5.6 Page Table Walk Policies

We considered two different policies for performing the page walk in case of a SLL TLB miss in a distributed system.

- Page Table at remote core: In this policy, the core which has the L2 slice for the virtual address performs the page walk and then sends the new translation as a response

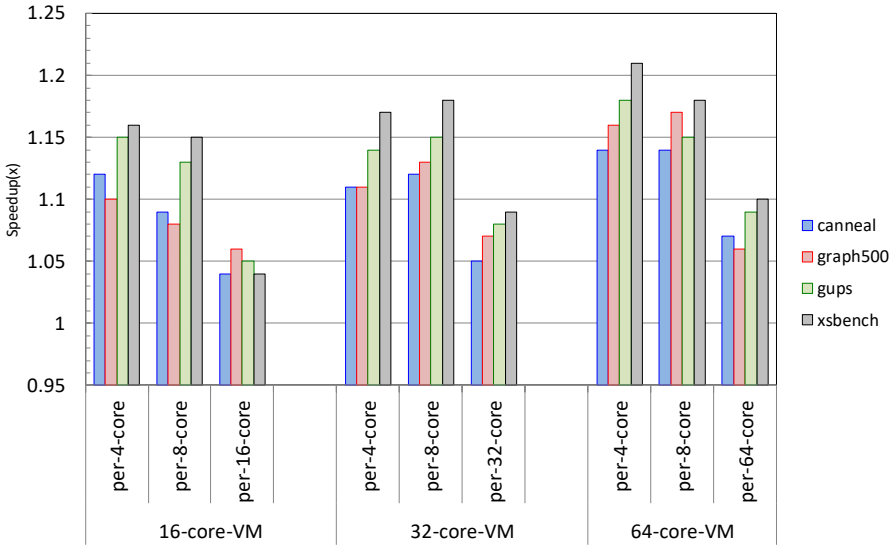

Figure 5.8: *Speedup of workloads comparing two different types of link acquisition policies*

to the requesting core after inserting it in the L2 slice.

- Page Table at requesting core: On a L2 TLB slice miss, a *miss* message is sent to the requesting core. The requesting core then performs the page table walk and sends an *insert* message to the remote slice.

Figure 5.9 shows speedup comparing the two types of page walk policies. While performing the page table at the remote node involves sending less number of messages in the interconnect, it pollutes the local cache of the remote core degrading its performance. Thus, we see that performing the page table walk at requesting core delivers slightly better results compared to page table walk at remote core.

5.7 Multi-programmed workloads

We also investigated the SLL performance benefits for multiprogrammed combinations of workloads. Our target platform is the 32-core Haswell system. Our workloads consist of four of the applications presented in all our results thus far. Since there are 11 of them, we study 330 combinations of 4 workloads. Each workload executes 8 threads, to fully utilize all 32 cores.

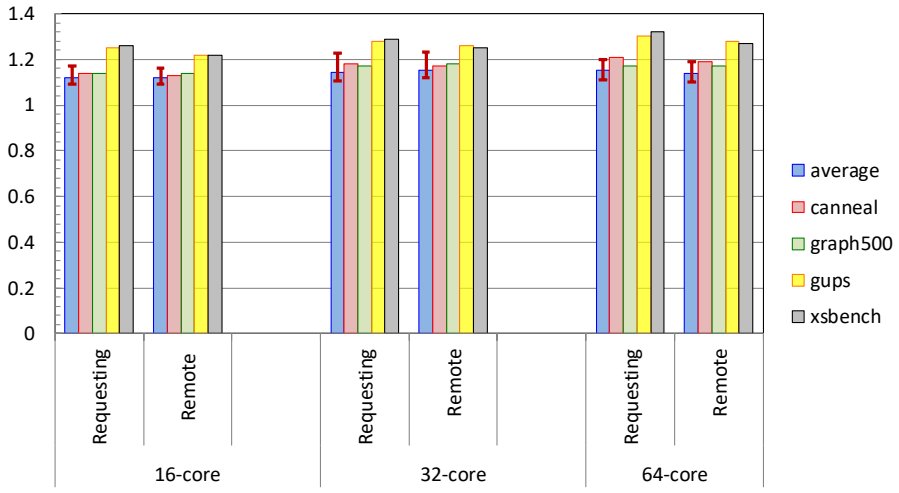

Figure 5.9: *Comparison between performing page walk at requesting core and remote core*

Figure 5.10 sorts the overall system throughput (IPC) improvement for our workloads. NUTRA is particularly effective for multiprogramming because it offers the utilization benefits of SLL versus PLL, without penalizing applications with high access latency. This explains why it *always* improves aggregate IPC compared to the other approaches, which can actually degrade IPC for a some workloads. In contrast, MONOLITHIC degrades performance for about half the workloads because of access latency issues while 10% of the workloads are degraded by DISTRIBUTED approaches.

To truly capture all aspects of fairness, we also show the speedup of the worst-performing application in our workload combinations in Figure 5.11. As shown, MONOLITHIC and DISTRIBUTED both see many workloads (almost half the combinations) where at least one application suffers performance loss due to high SLL TLB access latency. In some cases, the performance loss can be severe; e.g., 40% performance decrease. In contrast, only in 7% of the workloads does NUTRA see an application with performance loss. Not only is this relatively rare, the extent of the performance loss is relatively benign, with worst cases of 2-3% versus PLL TLBs. We believe that this problem is reminiscent of interference issues in LLCs and can readily be solved with traditional LLC QoS/fairness mechanisms [27, 28]. We leave these strategies for future work.

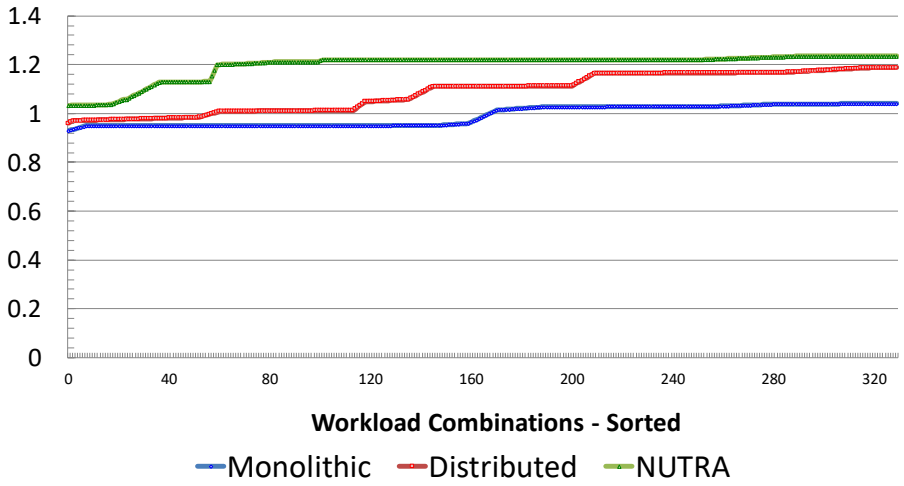

Figure 5.10: *Overall System throughput in 32 core architecture with 330 combinations of 4 workloads each*

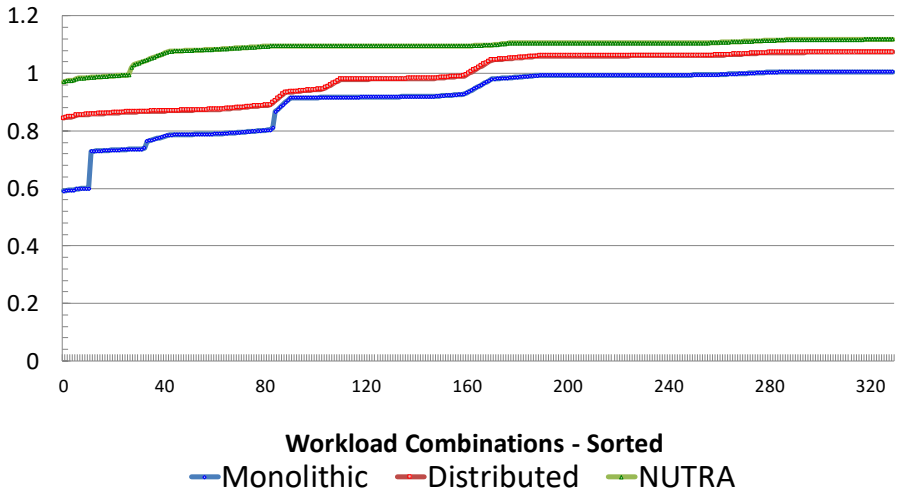

Figure 5.11: *Minimum speedup of workloads compared to a PLL TLB in a 32 core architecture*

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

Big-data based workloads have resulted in an increasing amount of stress on TLB hierarchy as address translation lies in the critical path of memory accesses. In multi-core architectures, this degrades performance mainly because of duplication in translations present in the private multi-level TLB hierarchies. While, a shared last level TLB has been shown to be beneficial, our experiments showed that a monolithic shared TLB structure leads to performance degradation because of the total latency involved in accessing an address translation.

The contribution of this thesis is a scalable approach to address translation in future architectures. In this chapter, we walk through some of the future directions and then conclude by summarizing the main contributions of this thesis.

6.1 Discussion and Future Work

6.1.1 Placement

NUTRA can be improved by adopting various placement [12, 29] and partitioning [30] techniques explored for NUCA architectures. Further, address translation pinning and migration on need can also be explored for NUTRA. Such optimizations can improve the maximum performance gains that NUTRA can achieve. I leave the performance improvements possible through these optimizations for future research.

6.1.2 Interconnect

Further, NUTRA mitigated network latency overhead by using a circuit-switched dedicated interconnect. Another approach to this problem could be to use the main network-

on-chip present in such CMPs. Using the main network for TLB access can complicate the translation request/response scheduling alongside other messages flowing in the network. This could be addressed by always prioritizing TLB requests over data requests. This is reasonable considering the high latency sensitivity but low bandwidth requirement for TLB requests as this work shows. Another possible optimization could be to replace the asynchronous path setup approach taken by NUTRA with alternate cycle selection and traversal, where selection of links happen in odd cycles and traversal takes place in even cycles.

6.1.3 Topologies

Further, NUTRA can also be extended to other topologies employed by CMPs. NUTRA utilizes a SMART based interconnect system for address translation packets to traverse throughout the system. Although we assumed a mesh based network among the cores for our evaluations, interconnects based on SMART have been shown to be favorable for all kinds of topologies.

6.1.4 Routing

In my evaluations, a routing policy of *XY* was assumed for traversals throughout the TLB interconnect. An *XY* routing policy is a turn-restriction based routing algorithms that relies on deadlock avoidance. As shown in Figure 6.1, *XY* routing restricts all Y to X turns and only allows X to Y turns. This is done in order to avoid deadlocks. Such a routing policy was used in the evaluations in order to keep the complexity low. NUTRA can adapt any kind of routing policy to further improve the speedup. We discuss the effect of contention on average latency in Section 5.4. The gap in performance observed between NUTRA and NUTRA-IDEAL can be further reduced by adapting a more relaxed routing policy.

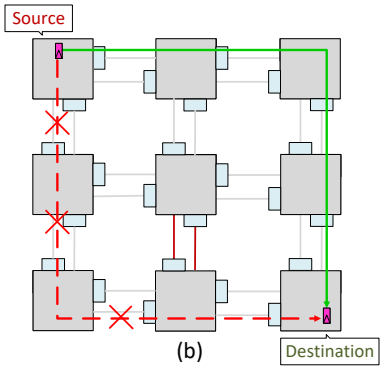

Figure 6.1: *An XY based routing policy does not allow Y to X turns*

6.2 Conclusions

In this work, I presented a scalable solution for the address translation needs in multi-core environment called NUTRA. NUTRA can be used to design last level TLBs in modern multi-processor architectures. This study demonstrates that taking a distributed approach like NUTRA for TLBs in CMPs is the way forward for performance gains. This work finds that the higher hit rate delivered by an SLL TLB does not translate to increase in performance. This is mainly because of two reasons: First, as we scale the number of cores, the required TLB size increases, increasing the access latency of the TLB structure. Secondly, the latency involved in traversing to the shared TLB structure through the traditional networks result in increase of total latency involved in translation. By employing a *co-design* of distributed architecture with a low-latency interconnect, I showed that NUTRA can deliver performance improvements in both multi-threaded and multi-programmed workloads.

REFERENCES

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08, Toronto, Ontario, Canada: ACM, 2008, pp. 72–81, ISBN: 978-1-60558-282-5.

[2] M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta, "The impact of architectural trends on operating system performance," *SIGOPS Oper. Syst. Rev.*, vol. 29, no. 5, pp. 285–298, Dec. 1995.

[3] T. W. Barr, A. L. Cox, and S. Rixner, "Translation caching: Skip, don't walk (the page table)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, Saint-Malo, France: ACM, 2010, pp. 48–59, ISBN: 978-1-4503-0053-7.

[4] G. B. Kandiraju and A. Sivasubramaniam, *Going the distance for TLB prefetching: an application-driven study*, 2. IEEE Computer Society, 2002, vol. 30.

[5] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Redundant memory mappings for fast access to large memories," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, Portland, Oregon: ACM, 2015, pp. 66–78, ISBN: 978-1-4503-3402-0.

[6] A. Saulsbury, F. Dahlgren, and P. Stenström, *Recency-based TLB preloading*, 2. ACM, 2000, vol. 28.

[7] A. Borg, J. B. Chen, and N. P. Jouppi, "A simulation based study of tlb performance," in *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, IEEE, 1992, pp. 114–123.

[8] M. Talluri and M. D. Hill, *Surpassing the TLB performance of superpages with less operating system support*, 11. ACM, 1994, vol. 29.

[9] P. Hammarlund, "4th generation intel x2122; core processor, codenamed haswell," in *2013 IEEE Hot Chips 25 Symposium (HCS)*, 2013, pp. 1–35.

[10] *www.amd.com*.

[11] A. Bhattacharjee, D. Lustig, and M. Martonosi, "Shared last-level tlbs for chip multiprocessors," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, ser. HPCA '11, Washington, DC, USA: IEEE Computer Society, 2011, pp. 62–63, ISBN: 978-1-4244-9432-3.

[12] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive nuca: Near-optimal block placement and replication in distributed caches," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 184–195, 2009.

[13] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *SIGARCH Comput. Archit. News*, vol. 30, no. 5, pp. 211–222, Oct. 2002.

[14] R. Ho, K. Mai, and M. Horowitz, "Managing wire scaling: A circuit perspective," in *Proceedings of the IEEE 2003 International Interconnect Technology Conference (Cat. No.03TH8695)*, 2003, pp. 177–179.

[15] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe, "The 48-core scc processor: The programmer's view," in *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[16] T. Krishna, C. H. O. Chen, W. C. Kwon, and L. S. Peh, "Breaking the on-chip latency barrier using smart," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 378–389.

[17] C. H. O. Chen, S. Park, T. Krishna, S. Subramanian, A. P. Chandrakasan, and L. S. Peh, "Smart: A single-cycle reconfigurable noc for soc applications," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 338–343.

[18] A. Bhattacharjee and M. Martonosi, "Characterizing the tlb behavior of emerging parallel workloads on chip multiprocessors," in *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, 2009, pp. 29–40.

[19] Y. H. Chen, T. Krishna, J. Emer, and V. Sze, "14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 262–263.

[20] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, ISBN: 0122007514.

[21] K. Sewell, R. G. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. F. Wenisch, D. Sylvester, D. Blaauw, and T. Mudge, "Swizzle-switch networks for many-core systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 278–294, 2012.

[22] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 126–137, Jun. 2007.

[23] C.-H. Chen, "Design and implementation of low-latency, low-power reconfigurable on-chip networks," PhD thesis, Massachusetts Institute of Technology. Department of Electrical Engineering and Computer Science., http://hdl.handle.net/1721.1/109002, 2017.

[24] *www.intel.com*.

[25] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, London, UK, 2012.

[26] V. Karakostas, J. Gandhi, A. Cristal, M. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Energy-efficient address translation," in *International Symposium on High Performance Computer Architecture*, ser. HPCA '16, 2016.

[27] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grained partitioning," in *International Symposium on Computer Architecture*, ser. ISCA '11, 2011.

[28] H. cook, M. Moreto, S. Bird, K. Dao, D. Patterson, and K. Asanovic, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *International Symposium on Computer Architecture*, ser. ISCA '13, 2013.

[29] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son, "A novel migration-based nuca design for chip multiprocessors," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, IEEE, 2008, pp. 1–12.

[30] H. Dybdahl and P. Stenstrom, "An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 2–12.

[31] S. Srikantaiah and M. Kandemir, "Synergistic tlbs for high performance address translation in chip multiprocessors," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43, Washington, DC, USA: IEEE Computer Society, 2010, pp. 313–324, ISBN: 978-0-7695-4299-7.

[32] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "Colt: Coalesced large-reach tlbs," in *International Symposium on Microarchitecture (MICRO)*, 2012.

[33] B. Pham, A. Bhattacharjee, Y. Eckert, and G. H. Loh, "Increasing tlb reach by exploiting clustering in page translations," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 558–567.

[34] A. Seznec, "Concurrent support of multiple page sizes on a skewed associative tlb," *IEEE Trans. Comput.*, vol. 53, no. 7, pp. 924–927, Jul. 2004.

[35] G. Cox and A. Bhattacharjee, "Efficient address translation for architectures with multiple page sizes," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17, Xi'an, China: ACM, 2017, pp. 435–448, ISBN: 978-1-4503-4465-4.